---

title: ERC-20 Token crowdsale standard

author: Frank Bonnet <?>,

type: ?

category: ?

status: Final

created: 22/06/2018

---

## Simple Summary

A standard interface to manage the crowdsale of an ERC-20 token.

## Abstract

The following standard allows for the crowdsale of an ERC-20 token on the Ethereum blockchain.

## Motivation

A standard interface for the crowdsale of an ERC-20 token allows any venture to raise funds in a decentralized manner on the Ethereum blockchain.

## Specification

## Crowdsale

### Methods

#### isInPresalePhase

This function shows if the crowdsale contract is currently in the presale phase.

``` js

function isInPresalePhase() public view returns (bool)

```

#### isEnded

This function shows if the crowdsale contract has ended.

``` js

function isEnded() public view returns (bool)

```

#### hasBalance

This function shows if there are ERC-20 tokens allocated to go towards the given Ethereum address at the end of the crowdsale.

``` js

function hasBalance(address \_beneficiary, uint \_releaseDate) public view returns (bool)

```

#### balanceOf

This function shows the amount of ERC-20 tokens that are allocated to go towards the given Ethereum address at the end of the crowdsale.

``` js

function balanceOf(address \_owner) public view returns (uint)

```

#### ethBalanceOf

This function shows the amount of ether in the given Ethereum address of the crowdsale investor that is allocated for the crowdsale transaction.

``` js

function ethBalanceOf(address \_owner) public view returns (uint)

```

#### refundableEthBalanceOf

This function shows the amount of invested and refundable ether for the given Ethereum address (only contributions during the ICO phase are registered).

``` js

function refundableEthBalanceOf(address \_owner) public view returns (uint)

```

#### getRate

This function shows the rate at which wei will be converted into ERC-20 tokens, based on the phase of the crowdsale and the amount of wei that is contributed.

``` js

function getRate(uint \_phase, uint \_volume) public view returns (uint)

```

#### toTokens

This function converts the amount of wei sent by the contributor into tokens using the `\_rate` from the getRate function.

``` js

function toTokens(uint \_wei, uint \_rate) public view returns (uint)

```

####

I’m not yet sure what this function does.

``` js

function () public payable

```

#### contribute

I’m not yet sure what this function does.

``` js

function contribute() public payable returns (uint)

```

#### contributeFor

I’m not yet sure what this function does.

``` function contributeFor(address \_beneficiary) public payable returns (uint)

```

#### withdrawTokens

I’m not yet sure what this function does. Where does it withdraw the allocated tokens from?

``` js

function withdrawTokens() public

```

#### withdrawTokensto

I’m not yet sure what this function does. Where does it withdraw the allocated tokens from? What’s the difference with the previous function?

``` js

function withdrawTokensTo(address \_beneficiary) public

```

#### withdrawEther

I’m not yet sure what this function does. Where does it withdraw the allocated ether from?

``` js

function withdrawEther() public

```

#### withdrawEtherTo

I’m not yet sure what this function does. Where does it withdraw the allocated ether from? What’s the difference with the previous function?

``` js

function withdrawEtherTo(address \_beneficiary) public

```

#### refund

This function refunds the amount of wei that was contributed, in the case of an unsuccessful crowdsale. The crowdsale is considered unsuccessful if minAmount was not raised before the end of the crowdsale.

``` js

function refund() public

```

#### refundTo

This function refunds the amount of wei that was contributed, in the case of an unsuccessful crowdsale. The crowdsale is considered unsuccessful if minAmount was not raised before the end of the crowdsale. How does this differ from the previous function?

``` js

function refundTo(address \_beneficiary) public

```

## Implementation

There are already plenty of ERC20-compliant tokens deployed on the Ethereum network.

Different implementations have been written by various teams that have different trade-offs: from gas saving to improved security.

#### Example implementations are available at

- https://github.com/OpenZeppelin/zeppelin-solidity/blob/master/contracts/token/ERC20/StandardToken.sol

- https://github.com/ConsenSys/Tokens/blob/master/contracts/eip20/EIP20.sol

#### Implementation of adding the force to 0 before calling "approve" again:

- https://github.com/Giveth/minime/blob/master/contracts/MiniMeToken.sol

## History

Historical links related to this standard:

- Original proposal from Vitalik Buterin: https://github.com/ethereum/wiki/wiki/Standardized\_Contract\_APIs/499c882f3ec123537fc2fccd57eaa29e6032fe4a

- Reddit discussion: https://www.reddit.com/r/ethereum/comments/3n8fkn/lets\_talk\_about\_the\_coin\_standard/

- Original Issue #20: https://github.com/ethereum/EIPs/issues/20

## Copyright

Copyright and related rights waived via [CC0](https://creativecommons.org/publicdomain/zero/1.0/).